AR-010-635

Attribute Café: A Java/CORBA Technology Experiment

T. H. Toh, M. P. Phillips and R. J. Vernik

DSTO-TR-0722

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Attribute Café: A Java/CORBA Technology Experiment

T.H. Toh, M.P Phillips and R.J. Vernik

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0722

ABSTRACT

This report describes an experiment which was undertaken to consider a range of software engineering issues associated with the development of component-based distributed applications. The experiment focused on the use of Java and Common Object Request Broker (CORBA) technologies as used in the development of a software visualisation demonstator application (the Attribute Café) which extracts, integrates and presents software-related information. This experiment has raised a number of issues that need to be considered if Java and CORBA are to provide basis for application development. These include architectural considerations, support for component-based development, technolology selection issues, design approaches, and the need for effective tool support.

RELEASE LIMITATION

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108

Telephone: (08) 8259 5555
Fax: (08) 8259 6567
Commonwealth of Australia 1998
AR-010-635
October 1998

Attribute Café: A Java/CORBA Technology Experiment

Executive Summary

The computing industry is developing new software technologies at an alarming rate. In order to apply these technologies effectively, research needs to be conducted to help understand and define the key characteristics of these technologies, the engineering tradeoffs that need to be considered, and the applicability of various development approaches. This report argues that the use of technology experiments can help address these requirements by undertaking controlled case studies for particular application domains of interest. The results of this work can then provide practitioners with the information that is required to effectively use the technologies and can provide developers with insights into how the technologies might best evolve.

Several new technologies are emerging to support the development of component-based and distributed systems. These include Java technologies such as JavaBeans and RMI, Microsoft technologies such as ActiveX and COM/DCOM, and the Common Object Request Broker Architecture (CORBA). Within the context of a technology experiment, this report discusses experiences gained in the application of Java and CORBA as part of an experiment which involved the development of a demonstrator software visualisation application, the Attribute Café.

The Attribute Café experiment has helped raise a number of issues that need to be considered if Java and CORBA are to provide an effective basis for application development. These include architectural considerations, support for component-based development, technolology selection issues, design approaches, and the need for effective tool support.

Although this work has provided useful results in terms of its original objectives, there are a range of other issues that need to be addressed in subsequent technology experiments. For example, this experiment considered the use of Java and CORBA technologies but did not make comparisons with competing technologies such as Microsoft's ActiveX and DCOM. Moreover, there is a need to consider how the technologies might co-exist if used in the same application and to assess performance characteristics. Although this work focused on the software visualisation domain, the results may be useful when considering the application of these technologies in other domains (e.g. Defence command and control information systems).

Authors

Leslie Toh
Information Technology Division

Leslie Toh is a researcher in Software Systems Engineering Group, Information Technology Division. His research interests include distributed systems technology, component system architectures and information acquisition from distributed sources. Leslie has a double Bachelor degree in Electrical & Electronic Engineering and Computer Science from the University of Adelaide.

Matthew Phillips
Information Technology Division

Matthew Phillips is a researcher employed in Software Systems Engineering Group, Information Technology Division. His research interests include distributed systems, programming language design, software visualisation and component-based software engineering. Matthew has a Bachelor of Computer Science (with Honours) from The University of Adelaide.

Rudi Vernik
Information Technology Division

Dr. Rudi Vernik is employed as a Principal Research Scientist and is Head of Software Systems Engineering Group. His research interests focus on the definition, development, and application of new systems and software engineering approaches to support Defence in its development of capabilities that will facilitate knowledge and information-based warfare. His group is currently undertaking research in the areas of system visualisation, component-based software engineering, systems characterisation and modelling, systems dynamics, and evolutionary capability development processes. Rudi has a Ph.D. in Computer and Information Science (Software Engineering) from the University of South Australia. He also has a Bachelor of Electronics Engineering (with Distinction) and a Diploma of Communications Engineering from the Royal Melbourne Institute of Technology.

Contents

Abbreviations

API	Application Programming Interface
AWT	Abstract Windowing Toolkit
CBSE	Component-Based Software Engineering
CM	Configuration Management
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSM	Composite System Model
DCOM	Distributed Component Object Model
GIOP	Generic Inter-ORB Protocol
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
ISB	Internet Service Broker
JCSE	Joint Command Support Environment
JDK	Java Development Kit
JFC	Java Foundation Classes
MVC	Model/View/Controller
OMG	Object Management Group
OO	Object-Oriented
ORB	Object Request Broker
RMI	Remote Method Invocation
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator

1. Introduction

Several new technologies are emerging to support the development of component-based distributed systems. These include Java technologies such as JavaBeans and RMI, Microsoft technologies such as ActiveX and COM/DCOM, and the Common Object Request Broker Architecture (CORBA). Although significant effort has been devoted to developing and marketing the technologies, far less effort has been directed towards understanding how they can be applied, their characteristics, tradeoffs, development approaches, and effective tool support. This report argues that technology experiments (a type of controlled case study) need to be undertaken for various application domains to study and report on these software engineering issues. Within the context of a technology experiment, this report discusses experiences gained in the application of Java and CORBA technologies as part of an experiment involving the development of a demonstrator software visualisation application, the Attribute Café.

Attribute Café provides a basis for accessing a range of diverse sources of software engineering information, for extracting attribute information relating to software entities (e.g. source files), and integrating this information into a consolidated form view. Other integrated views are provided which provide alternative views for selected parts of the main view, such as charts (for numeric information) and text views (e.g. for source code). The specific goals of the experiment were to:

1. Study aspects of Java and CORBA, including implementations, design approaches, tool support, and technology integration, in order to understand the key characteristics of these technologies and determine how they might best be applied to application development.
2. Gain practical experience with Java and CORBA to determine the tradeoffs and key issues in using these technologies.
3. Assess how a Java/CORBA approach could be used as the basis of a family of proposed Integrated Visualisation Environments (Vernik 1996)

The report is structured as follows: Section 2 provides some background on the experimental context. It provides details of the application domain and gives an overview of the Attribute Café demonstrator. Section 3 discusses the technologies and approaches that were studied, including Java/CORBA integration, the design approach, and the implementation approach. The results of this technology experiment are then discussed in Section 4. Some of the results include architectural considerations, the use of supporting tools support, component-based development issues, and technology implementation/selection issues. Conclusions are provided in Section 5.

2. Experimental Context

2.1 Software Systems Visualisation Research

Information Technology Division has been conducting research into aspects of software systems visualisation and description in order to provide more effective visibility and better management of large Defence software systems. A key outcome of this research has been the Integrated Visualisation and Description Approach (Vernik 1996) which allows information from diverse project information sources to be accessed, integrated, and customised to support the specific needs of users. The approach is based on the use of a Composite Systems Model (CSM) which provides the basis for accessing and integrating information as well as supporting the generation of integrated computer-based visualisations of the software system. The CSM comprises the structural entities of the software (e.g. design elements, modules, code) and their relationships. It also includes key attributes of the entities. For example, in terms of a model of a software system, the CSM would include the set of software entities, the relationships between them and a set of attributes for each of the entities (eg who wrote a particular module, test coverage, configuration status, product metrics etc). Computer-based visualisations can then be generated to provide multi-perspective views of the system which focus on the main aspects of interest to a user.

In developing practical implementations of the Integrated Visualisation and Description Approach, there are a number of issues that need to be considered, including:

- Approaches for accessing and integrating diverse, distributed data sources;
- Architectural issues related to the design and development of integrated visualisation environments; and
- The use of component-based software engineering approaches/tools to support the development and use of component assets and 'pluggable' architectures.

The new component and distributed systems technologies show promise for addressing many of these issues. The Attribute Café demonstrator, although simple in nature, provides a vehicle for exploring some of these issues in terms of Java and CORBA technologies.

2.2 Overview of Attribute Café

The Attribute Café system consists of a *client*, a *server* and any number of *data sources*, each occupying a different tier in a three-tier client/server model. Figure 2-1 depicts this model, showing the tier in which each of the components reside and how they relate to each other. The server, which makes up the Middleware Layer, connects to each of the data sources comprising the Data Layer and unifies the data provided by each source into a single logical data source. The client, which occupies the Client Layer, connects to the server and is unaware of the Data Layer's structure. This architecture allows data sources to be added, removed, changed or moved to new locations without affecting the client.

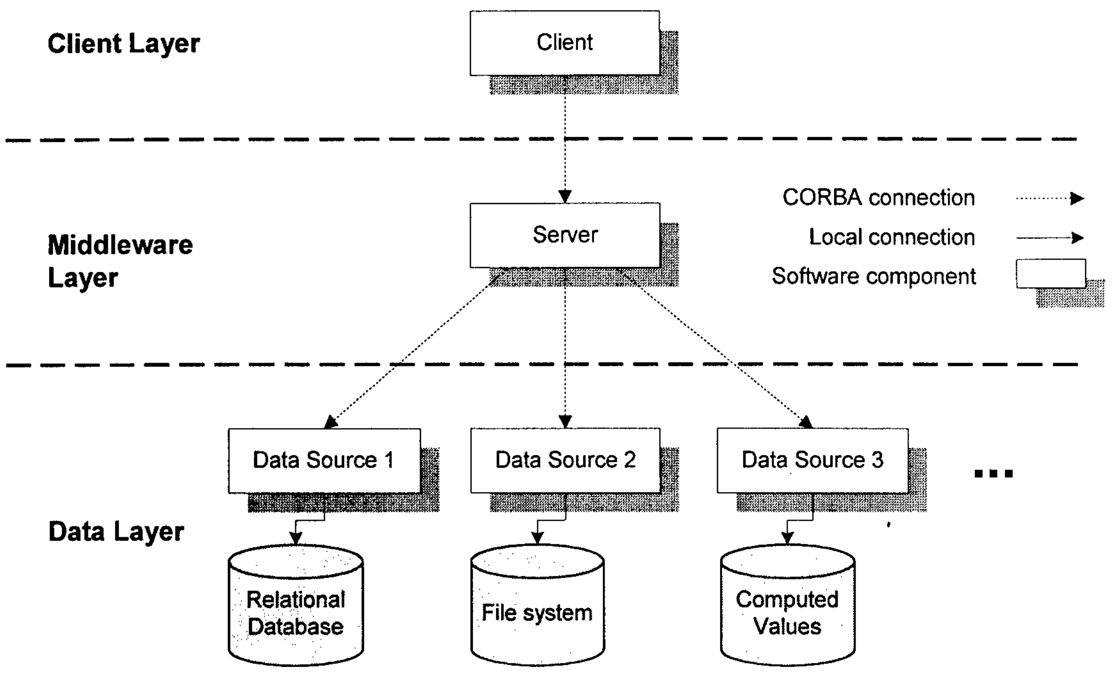


Figure 2-1 General Concepts of Attribute Café Architecture

Each of the Attribute Café data sources provides a different set of software attributes, and while each data source presents the same interface to the server, the actual attribute values it provides might be generated in a variety of ways. In Figure 2-1 for example, three example data sources are shown, one reading from a relational database, one reading from a file system and one generating data from computations based on other values.

The actual data sources used in the Attribute Café demonstrator are simulations of real data sources that may be present in a software project. The data sources are:

- **File Properties**: extracts data from file system (e.g. file creation date).
- **Configuration Management (CM)**: extracts data from the project's configuration-management system (e.g. file author).
- **Build System**: extracts data from the project's build system (e.g. module name).
- **File Metrics**: generates metrics from physical attributes of the source code (e.g. line count).
- **File Source**: generates URL's to access the contents of each source file.

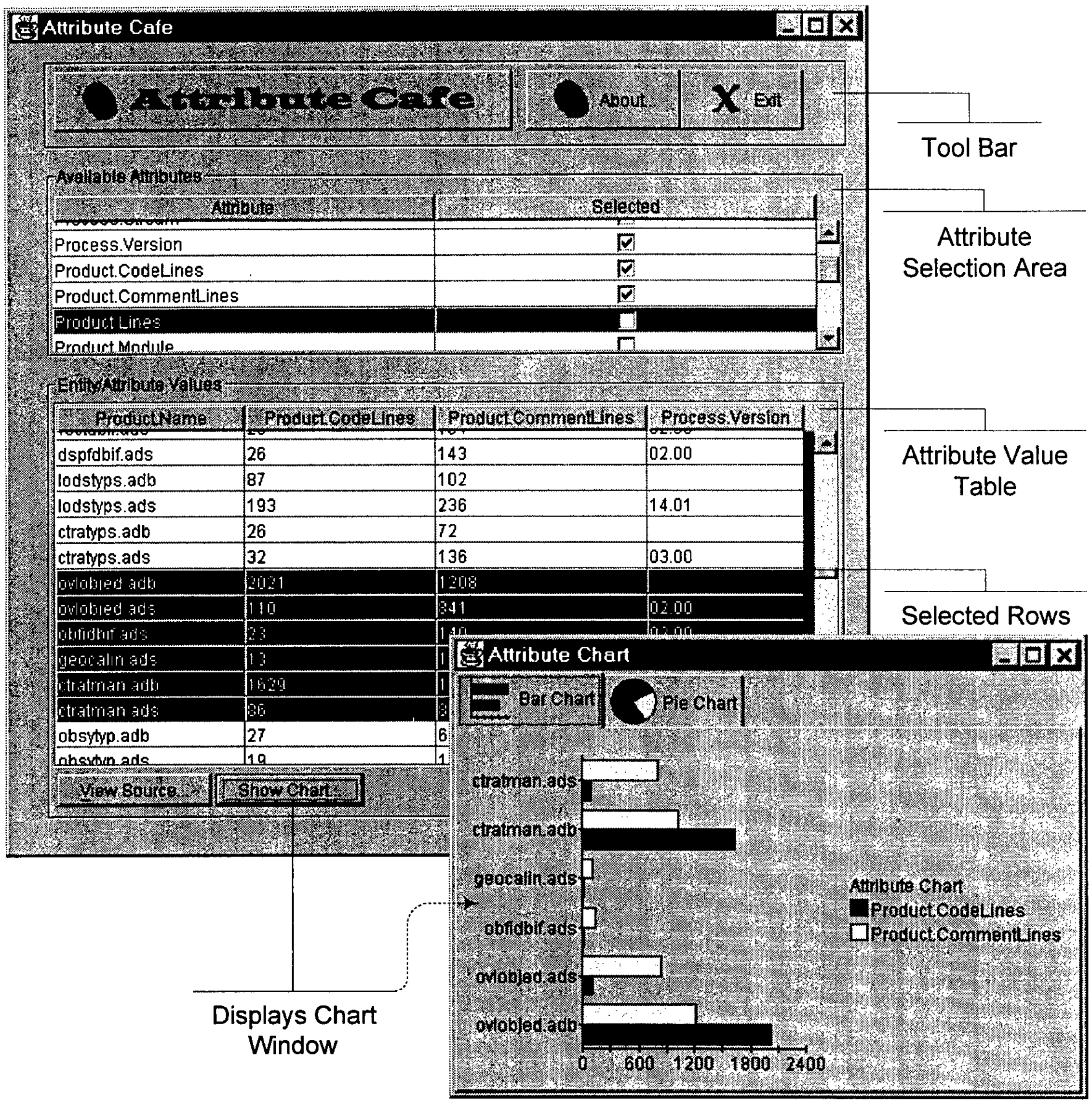
The attributes provided by each of these five data sources are listed in Table 2-1. The attributes have been broken into three categories as defined by (Fenton, Pfleeger et al. 1994): *Process, Product* and *Resource*. These are logical categories and are not necessarily related to the attributes' physical sources (for example, the CM Data Source provides attributes in both the *Process* and *Resource* categories). Process attributes are those relating to the software engineering process and might include source code version numbers and completion status. Product attributes come from a physical product of the project (eg source code) and include information like file name, file size and the date file was last modified. Resource attributes indicate what resources are associated with each entity (eg author, time allocated).

Table 2-1 Attribute Café Data Sources and Attributes

	Attributes		
Data Source	Process	Product	Resource
File Properties	LastModifed	Name Path Size Type	
CM	Version		Analyst Author
System Build	CI Stream	Module	
File Metrics		CodeLines CommentLines Lines	
File Source		Source.HTML Source.Text	

The example data supplied by the five data sources was generated from the source code release of a large Defence command & control information system (the Joint Command Support Environment (JCSE) Release 4). There are 3,057 entities in the example model, each with fifteen attribute values giving a total of 45,855 values available from the Attribute Café server. In this experiment, all data provided by the data sources are stored in five relational databases, but we have also experimented with data sources which obtained data from diverse sources such as local file systems and web servers.

The Attribute Café client graphical interface is shown in Figure 2-2. This interface consists of a main window which presents data from the server in two tables. The complete list of attributes provided by the server is displayed in the topmost 'Available Attributes' table, which allows the user to select the attributes that they want to be shown in the 'Entity/Attribute Values' table below. The attribute values table displays one row per entity, with each column containing the associated attribute values for the entity. In this example, the user has displayed the *Product.Name, Product.CodeLines, Product.CommentLines* and *Process.Version* attributes.



Attribute	Selected
Process.Version	✓
Product.CodeLines	✓
Product.CommentLines	✓
Product.Lines	☐
Product.Module	☐

Product.Name	Product.CodeLines	Product.CommentLines	Process.Version
dspfdbif.ads	26	143	02.00
lodstyps.adb	87	102	
lodstyps.ads	193	236	14.01
ctratyps.adb	26	72	
ctratyps.ads	32	136	03.00
ovlobjed.adb	2021	1208	
ovlobjed.ads	110	841	02.00
obfidbif.ads	23	140	02.00
geocalin.ads	13	1	
ctratman.adb	1629	1	
ctratman.ads	86	8	
obsytyp.adb	27	6	
obsytyp.ads	19	1	

Figure 2-2 Attribute Café Client Interface

The Attribute Café client also allows numeric attribute values to be displayed graphically with a charting tool. The chart in Figure 2-2 was created by selecting the six entities (highlighted rows) and then clicking the 'Show Chart' button. In this example only the *Product.CodeLines* and *Product.CommentLines* attributes are numeric, so these values have been charted as two bars against the *Product.Name* attribute of each selected entity. Attribute values can either be displayed as a bar graph (as in Figure 2-2) to compare absolute values or as a pie chart to compare scaled values (such as when the attribute values represent a percentage).

The source text for each entity may be displayed by selecting the entity and clicking the 'View Source' button. The source code is accessed via an HTTP connection to a web server. The resulting text is displayed either in a new browser window (if the client is running as an applet) or in a custom viewer that uses a lightweight HTML Java component (if the client is running as an application).

3. Technologies and Development Approaches Used

3.1 Use of CORBA

The Common Object Request Broker Architecture (CORBA) (Vinoski 1997) is an emerging open distributed object computing infrastructure standardised by the Object Management Group (OMG) (OMG 1997). CORBA provides an object-oriented distributed system infrastructure on which a heterogeneous set of software applications can transparently interact with each other through well-defined interfaces. Client objects can issue requests to server objects which perform services on their behalf. The implementation and location of a server object is hidden from the requesting client.

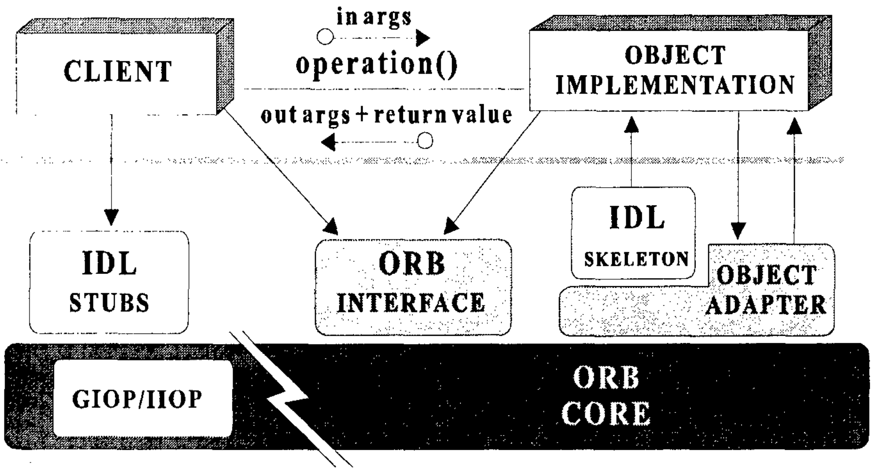


Figure 3-1 CORBA Architecture

Figure 3-1 depicts a simplified picture of how a client and a server interact. The interfaces through which object communication is performed are described by the OMG's Interface Definition Language (IDL). The IDL is mapped to standard programming languages, including Java (Orfali and Harkey 1997), for the implementation of applications.

The CORBA stub on the client side helps 'marshal' a request as a message and routes it to the platform that contains the server object. The Object Request Broker (ORB) provides a mechanism for transparently communicating client requests to target object implementations. The CORBA skeleton on the server then translates this message (or 'de-marshals') to a form that the target server object can understand. An implementation repository (on the server side) and an interface repository keep track of all IDL-defined interfaces.

An ORB is a logical entity that may be implemented in various ways, such as one or more processes or a set of libraries. To decouple an application from its implementation details, the CORBA specification defines an abstract ORB interface. The Object Adapter assists the ORB with delivering requests to the server object and with activating the server object. More importantly, an Object Adapter associates object implementations with the ORB. Communication between the ORBs at disparate locations may be vendor-specific. However, the specification for the second version of CORBA (CORBA 2.0) defines a common protocol known as Generic Inter-ORB Protocol (GIOP) for communication between ORBs from different vendors. An Internet implementation of this protocol, the Internet Inter-ORB Protocol (IIOP), is supported in ORBs from most vendors.

A wide range of CORBA-compliant products are currently available, including Iona's Orbix and OrbixWeb (Iona 1997), IBM's Component Broker (IBM 1997) and Visigenic's[1] Visibroker (Visigenic 1997). There are also a number of free CORBA products, including Sun's JavaIDL (Sun 1997) and Xerox's ILU (Xerox 1997). All of these ORBs provide basic CORBA features and most provide interoperability with other ORBs via IIOP. A number of vendors are also starting to offer CORBA services, such as the Event Service, Transaction Service, Naming Service, etc. Additionally, many commercial ORBs provide, or will provide, some level of integration between CORBA and Microsoft's DCOM.

After conducting a survey into the CORBA products available, we decided to use the Visigenic Visibroker for Java ORB. A number of factors contributed to the decision to use Visibroker as the ORB for Attribute Café. Foremost was the level of support from vendors such as Inprise and Netscape, which has integrated the Visibroker Java client libraries into Netscape Communicator 4 and its Enterprise Server 4 package. The Visibroker ORB is also written in Java, allowing both servers and clients to run anywhere there is a Java Virtual Machine. Additionally, Visibroker provides enhanced Java support through the use of its Caffeine technology which allows any client/server interface to be specified in Java and then automatically converted into CORBA IDL. The Caffeine extensions in the Visibroker ORB also allow Java objects to be transparently passed by value, a feature that is not yet available in CORBA.[2] Section 3.3 contains further discussion on the use of Caffeine.

By using ORB from a single vendor, we have chosen not to investigate the issue of interoperability of ORBs between different vendors as part of the experiment. Given that CORBA interoperability is an independent issue that is well researched and being showcased by CORBAnet (http://www.corba.net), the simplification of the experiment is justified in our view. Instead of using the standard Visibroker for Java ORB, initially we opted to experiment with Netscape Enterprise Server 4 that integrates Visibroker 2.5 - known as Netscape Internet Service Broker (ISB) - with its web server. The aim of the exercise is to evaluate the Netscape Server as part of the technology experiment. A few compatibility problems were encountered during application development and while these problems may have been due to our lack of experience with ISB, we decided to switch to the standard Visibroker for Java 3.0 package for the remainder of the project. This solved our problems and did not affect the compatibility of Attribute Café with the Netscape Enterprise Server or Navigator ORB infrastructure. Section 4.4.2 describes these problems among other issues.

1 Inprise acquired Visigenic in early 1998.

2 An object-by-value extension to CORBA 2.0 is now part of CORBA 3.0, outlined at Comdex/Enterprise 98 by OMG.

3.2 Use of Java

Java is an object-oriented (OO) language containing features from C++ and several other OO languages. It was designed to be simple, robust and secure. Java's robustness derives both from its simplicity as language (which helps reduce programmer errors) and its memory-management scheme which precludes the use of traditional pointers and mandates automatic memory management (*garbage collection*). Security is achieved through the provision of a *security manager* component which polices Java's interactions with the environment and vetos those that could be harmful.

Java was also carefully designed to have no dependencies on a particular operating system/hardware platform combination.[3] This platform-independence is achieved by distributing compiled Java applications in *byte-code* form that is executed by a *Java Virtual Machine* on a particular platform. A comprehensive set of standard class libraries provides a common interface to platform services, such as *java.net* for Internet communication, and *java.awt* for the user interface, etc.

A relatively new addition to the Java platform is the JavaBeans component model, which allows the development of interoperable Java components. The JavaBeans model specifies how components provide their properties and events and how other components can dynamically discover these properties. One of JavaBeans' most important characteristics is that it is very 'light weight' – an object only need follow a simple set of design patterns to be JavaBeans compliant (most of the classes provided in the standard Java class libraries are now JavaBeans). Although a JavaBean may be very simple, a sophisticated JavaBean can provide many properties for customisation, events for notification of various changes and a customisation 'wizard' to streamline development.

Although Java applications can be developed using just a text editor and the command-line tools provided by the Java Development Kit, this is not an efficient way to build GUI elements or applications that must integrate many components. Borland's JBuilder was used to develop Attribute Café because it provides a flexible dual approach to Java development: a GUI builder tool for the user interface, and an integrated editor/class browser for code development. JBuilder also aids the component-based development approach by managing a repository of JavaBeans that can be 'dropped' into a project and connected together. It also provides a number of 'wizards' that automate common development tasks, such as generating classes from predefined templates and packaging existing classes as JavaBeans. Additionally, JBuilder automatically manages the build/compile/deploy process, which can be complex for Java applications that typically have many classes, packages and associated resources.

JBuilder provides a large number GUI JavaBeans, but it was decided early on that the Attribute Café client would use the Java Foundation Classes (JFC) Swing Set (JavaSoft 1997) wherever possible. This was decided for a number of reasons, the primary ones being to test the use of a third-party component set with JBuilder and

[3] This goal derives from Java's original intended use as in embedded systems environments such as cable set-top boxes where many hardware/OS combinations are used.

to become familiar with the next generation of the Java Abstract Windowing Toolkit (AWT).[4] The JFC also defines a flexible framework for its controls, enabling a high degree of integration with other components. It was also decided that the JClass charting Bean would provide the charting functionality needed to the bar and pie charts available in the client. Section 4.3 contains a comprehensive overview of the components used in Attribute Café.

3.3 Integrating Java and CORBA

As mentioned in Section 3.1, there is a standard language mapping between Java and the OMG's IDL. The traditional way of implementing a CORBA application is to specify the object interfaces with IDL, and then to use an IDL compiler to generate the necessary CORBA stubs and skeletons. This can be a source of inefficiency, since interfaces often change during development and, without support from development tools, software developers have to make manual changes to the implementations to reflect the change in interfaces. The two-step change process can become tedious and even overwhelming for large software applications where there are multi-level dependencies between object interfaces. As the Java language directly supports interface definition, it might be desirable to eliminate or automate the IDL writing stage and generate IDL from Java instead of vice versa.

The Voyager Core Technology from ObjectSpace (ObjectSpace) supports one such type of close Java/CORBA integration. As Voyager is primarily a Java agent-enabling system with CORBA integration,[5] all the CORBA communication is handled by the agent and hence no stubs or skeletons are needed. Voyager also claims to be able to process existing IDL interfaces to CORBA services, automatically create corresponding Java interfaces, and then use these interfaces to communicate with these services using natural Java language syntax.

Another option is a set of features collectively known as Caffeine (Visigenic 1997), which are incorporated in Visigenic Visibroker. Like Voyager, Caffeine provides tools to directly generate stubs and skeletons from Java interfaces. In the case where IDL is still required (e.g. for use with other languages), Caffeine can also generate IDL from Java interfaces. Caffeine extends the Java/CORBA language mapping to include any Java object class which has characteristics that do not conform to the rules in the standard mapping (Visigenic 1997). These objects are automatically passed by value using Java's serialisation mechanism.

Several products such as Netscape Enterprise Server and Inprise JBuilder provide Caffeine support through their incorporation of the Visibroker ORB into their products. We experimented with the main features of Caffeine to aid the development of Attribute Café, and Section 4.4 describes our experiences with the technology.

[4] JFC will replace the current AWT and become the standard user interface framework when Java 1.2 is released in late 1998.

[5] CORBA support was a new feature in the Beta 1 release of Voyager 2.0.

3.4 Object Oriented Design

The Unified Modelling Language (UML) (Rational 1997) is becoming the defacto standard for object-oriented and distributed system design. It is currently supported by a large number of tools including Microgold WithClass (Microgold 1997) and Rational Rose (Rational 1997). Many of these tools support automatic code generation from their UML models and can also 'reverse-engineer' code back into the original model, keeping the design in synchronisation with the implementation. This process is often called *round trip engineering* and may be performed a number of times during a project's development phase.

In order to facilitate the development of Attribute Café, and to perform an evaluation of the round trip development approach on a small project, the key classes used in Attribute Café were originally specified in UML using Rational Rose. The Rose code generation tool was then used to quickly generate Java templates, which were completed in the JBuilder environment. At several times during development, and at the completion of the project, the Attribute Café source was reverse-engineered back into the original model and the diagrams showing the classes and their relationships updated. During development, this model enabled the developers to maintain a common frame of reference, and now that the project is complete it provides the main design overview. A discussion of some of our findings while using Rose is contained in Section 4.2

The two main UML class diagrams developed for Attribute Café are shown in Appendix A. These diagrams represent the state of the Attribute Café client and server at completion of development. The *Server* interface (Figure A-2) is the key to the Attribute Café design, since it defines the sole communication interface between the client and the server.

The client GUI is managed by *ClientPanel* which extends the JFC *JPanel* class. *ClientPanel* has an associated *Server* instance and manages an *AttributeTableModel* (a JFC *TableModel* implementation) which makes selected attribute values read from the *Server* accessible to a JFC *JTable* (the 'Entity/Attribute Values' table in Figure 2-2). The *SelectedAttributesTableModel* allows the attributes, which are selected to be visible in the *AttributeTableModel,* editable in the 'Selected Attributes' table. The *ClientPanel* also uses a *HtmlFrame* to display source code and a *ChartFrame* to produce charts.

The server consists of the *ServerImpl* class that extends the *_ServerImplBase* class, which is the skeleton class automatically generated by Caffeine from the *Server* interface. *ServerImpl* maintains connections to the five *DataSource*'s which provide the actual data that the server unifies into a single logical source for the client. The *FileSource, CM, FileProp, FileMetrics* and *SystemBuild* classes provide the actual implementation of the five different data sources.

3.5 Implementation

The client and server were developed independently once the server interface was designed. This was done to help partition the work, and because the two applications share little in implementation technology. The client is essentially a GUI written in Java, making use of JavaBean component technology, while the server is mainly concerned with accessing and gathering database information. This approach also tested how well the two sides integrated after being developed virtually in isolation.

The client was implemented so that it can run both as a stand-alone application or as an applet in a web browser. This required that some features be aware of where the client is running in order to avoid applet security violations and to sensibly handle the display of HTML documents.[6] The client was also designed so that it could run with either the Visibroker CORBA stand-alone client library or the library provided by Communicator 4.

The fact that the client is unaware of the location and implementation of the server allowed the client to be initially tested using a 'dummy' local server which provided a small number of hard-wired attribute values. When the client/server combination was ready for testing, the dummy server was simply replaced by the real server. We were pleasantly surprised when the client and the server worked together the first time they were integrated.

Although each of the software components depicted in Figure 2-1 maps to an equivalent Java object, only the server object was implemented as a CORBA object at first. The data source objects were merely local Java objects managed by the server. Therefore, the initial implementation merged the middleware and data layer into one, creating a traditional server. The data sources were only made into CORBA servers after they had been fully implemented and tested. This was done to test the impact of making a distributed client-server system out of a local one. We found that only a few minor changes to the source code were required to achieve full distribution of the data source objects.

4. Results

4.1 Architectural Considerations

As shown in Figure 2-1, we adopted a relatively simple client-server architecture as a basis for the implementation of Attribute Café. The main departure from the traditional two-tiered architecture is the splitting of the data layer, and leaving the so-called 'business logic' functions to the server component. In Attribute Café's case, the business logic function of the server was to unify all the information from the various data sources and present it to the client. Information flow in this architecture is only one-way, from the data sources to the server and from the server to the client. The client is always the component that initiates CORBA requests.

The current architecture has the advantage of being simple to implement since there is only one physical server object that the client need recognise. However, as the number of clients increases, the single-threaded server can become a bottleneck. This is because standard CORBA requests are synchronous, meaning client requests will block until the server is ready to process them. With many clients connected to the

[6] When running as an applet, HTML documents are displayed using the host browser, but when running as an application, documents must be displayed in an HTML frame managed by the client

same server, most of them will spend more time waiting for the server to process their requests than doing useful work. Another disadvantage with the existing design is that the available data sources must be fixed at compile time since there is no provision for run-time modification to the data source pool.

Instead of using the standard synchronous requests, CORBA allows asynchronous requests which are non-blocking. In this scenario, the server issues a callback to the client once the operation is finished. This would alleviate the performance problem since it allows clients to continue accepting user input while waiting for the server. However, to address all the problems discussed, other architectural patterns need to be considered.

A factory-based architecture as shown in Figure 4-1 is a solution that addresses the problem. A client will request a dedicated server object from the *server factory* which creates and manages a pool of such server objects. Each server object is then dedicated to one particular client and behaves like the server in the single server model of Figure 2-1, until it is released. Thus the increase in client numbers is supported by adding more server objects to the resource pool, resulting in a more scalable architecture. The server factory also serves as a point of contact for all the data sources, and allows sources to be dynamically added, removed and relocated.

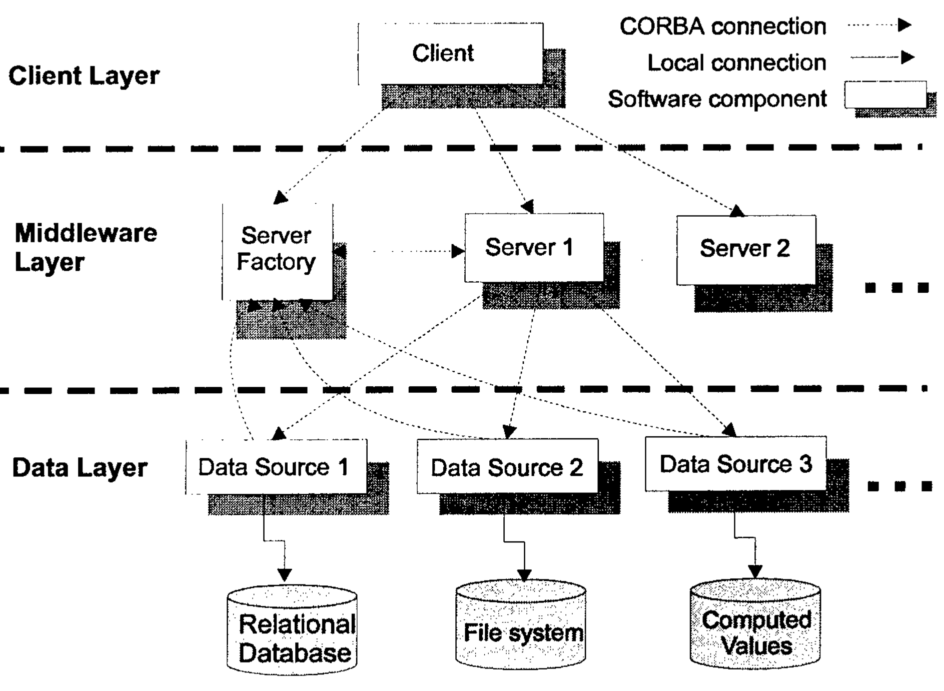


Figure 4-1 Factory-Based Architecture

Implementing this architecture would involve minimal changes to the client while solving the problems discussed in this section. The ideal of keeping a simple client in a three-tiered architecture is also preserved.

4.2 Tool Support

Although Rose proved to be a very useful tool in the development and documentation of Attribute Café, we found that in practice only one cycle of the round trip engineering process was possible. This was due to the fact that Rose requires 'marker' comments in the source in order to preserve non-UML information across the reverse-engineer/code generation process. If these marker comments are not present in a source file, which commonly happens when JBuilder generates the file, the code generation step will overwrite vital portions of the code with empty placeholders. This highlights an important issue with the integration of OO design tools with products from other vendors. Many newer OO tools claim not to require source markers, eg. Graphical Designer Pro (ASTI 1998) and Object Engineering Workbench (Software 1998).

JBuilder provides a number of important productivity features termed 'wizards'. One that was particularly useful in the development of Attribute Café was the 'implement interface' wizard, which helps in implementing a Java interface by automatically generating stubs for all interface methods. This wizard was used to help implement at least four interfaces, saving time and reducing errors. Other wizards allow selective overriding of inherited methods and automatic generation of JavaBeans from simple classes. As object-oriented/component-based application development becomes more common, this sort of intelligent tool support will greatly reduce the amount of work required to build robust applications.

It was recognised that some form of version control system would be needed in order to create regular snapshots of the source and to manage the situation where more than one developer attempted to modify the same file at once. The Revision Control System (RCS) (Tichy 1991) was chosen because it is freely available, easy to use and runs on a range of platforms. However, JBuilder provides no support for RCS, which made management of version control somewhat awkward. Consideration needs to be given to other configuration management solutions, especially in regard to their support for managing diverse development artefacts such as design models, documents and images. It is likely that in future projects the PVCS (Intersolv 1997) version control system will be considered, as it integrates with JBuilder and can manage heterogenous collections of information on a range of platforms.

4.3 Component-Based Development

A significant part our research is to investigate the use of a component-based development approach to software development, or Component-Based Software Engineering (CBSE). The CBSE approach embraces the 'buy, don't build' concept, and involves the creation of software systems from reusable components, as many as possible of which are purchased and not developed from scratch. The success of CBSE is therefore dependent on there being a sufficiently large pool of quality components available to meet project needs. In developing Attribute Café using a CBSE approach, we found that although Java and JavaBeans are still emerging technologies, there were sufficient JavaBeans components already available to meet our needs. The use of components greatly reduced the amount of time needed to complete the project and contributed to the understandability of the code.

Many components were used to build Attribute Café, including a number of the standard Java 1.1 Beans (eg *java.net.URL*), the prerelease JFC Beans and some of the third-party GUI support beans provided by JBuilder. Appendix B contains a complete list of components used in Attribute Café. One area where component-based development support was lacking was with the use of CORBA. This is expected to be remedied in the JBuilder Client/Server suite.

JBuilder's support for JavaBeans allowed the pre-release JFC Swing beans to be installed into the component palette and used within the Attribute Café project in the same way as the standard beans that are supplied with JBuilder. The JFC beans could be configured and previewed with the JBuilder UI designer, reducing the need to consult API documentation and access the Java classes directly. The JavaBean standard also helped ensure that the JFC beans interoperated correctly with beans supplied by other parties (such as the JClass components).

The charts displayed in the Attribute Café chart window are generated by the JClass *JCChart* bean, which can produce a wide variety of customisable charts displaying data from different data sources. *JCChart* is a complex component, yet it does not provide a custom property editor to help customise itself within JBuilder. This forces the user to become familiar with its API before being able to create even relatively simple charts. The interface that is provided for making data from custom sources is also counter-intuitive, and trial-and-error testing was required before it became clear how the interface methods were expected to function. This clearly highlights that, to be fully effective, a complex component needs to provide intelligent customisation tools and use interfaces that are clear and simple.

An important benefit was gained from the adoption of the Model/View/Controller (MVC) paradigm in the JFC and JClass packages (Gamma, Helm et al. 1995). In an MVC system, data is encapsulated by a *model* object to which any number of *views* can connect to provide visual representations of the model. A *controller* object is responsible for receiving user input from the views and updating the model appropriately. Whenever a change in the model occurs, all connected views are notified so they may update their displays. The main benefits of this model are that any representation may be chosen for data in the model and that more than one view/controller may display/edit the model simultaneously.

As a concrete example, the JFC *JTable* control (which is both view and controller) has an associated model defined by the *TableModel* interface. Thus, in order to display attribute values from the server in a *JTable*, it was only necessary to develop a class which provides a *TableModel* interface to data read from the server. Only the data displayed in the visible portion of the table is read from the server: a necessary requirement since there may be millions of values available from the server. It also allows more than one table to connect to a single *TableModel*, leaving open the possibility of adding advanced features such 'split view' which would let the user view several areas of the *TableModel* simultaneously.

4.4 Technology Implementation/Selection Issues

4.4.1 General Comments

Most of the technologies that we made use of in this experiment were still in their infancy. CORBA 2.0 only emerged as a standard in 1996, and Java in the similar time frame. The incorporation of Java/CORBA mapping into the CORBA standard and the subsequent support of commercial products are even more recent events. Products that were used in the development of Attribute Café like Netscape Enterprise Server 4.0, JBuilder and Visibroker 3.0 are either first or second generation software tools with little time for refinement since their initial release.

It is therefore a little surprising that we have not encountered significant obstacles in using these new tools and technologies. Some of the implementation approaches described in Section 3.5, such as the development of the client and server in isolation, and the conversion of the data source objects into CORBA servers were successful on the first try. Apart from some initial configuration problems, the experimental system was operational once the tested components were put together.

However, it has to be noted that with a single client and server system, we have not stressed the system enough to obtain a clear picture of how well such a distributed system performs under load. Larger scale experiments need to be conducted to investigate the overall performance and reliability of Java/CORBA based system.

4.4.2 Compatibility

One other aspect that warrants some discussion is the use of Caffeine technology in this experiment. We have used the tools that come as part of Caffeine in both Visibroker for Java 3.0 and Netscape Enterprise Server 4.0. It was found that the Netscape version of Visibroker, being the previous version, had some problem supporting non-standard complex data types. One such type that was used was the array of string type *String[]*. It appears that the tools Netscape supplied for Caffeine failed to generate all the necessary helper and container classes to support the passing of such types in CORBA. Visibroker 3.0, however, did not have the same problem.

We also encountered some difficulties when using the generic Java object class *java.lang.Object* as a return value from CORBA servers. Because there is no standard Java mapping to an equivalent CORBA object, Caffeine generates non-standard helper and container classes for it. The problem is that those helper classes were placed in the *java.lang* package, which is part of the Java standard framework. The addition of foreign classes to any *java* package is (correctly) vetoed by the Java applet security manager and thus prevented the Attribute Café client from running in a browser. The classes had to be moved to a new package manually to get over the problem. We have decided to continue using the non-standard classes because they are simpler to manipulate than the standardised generic CORBA object *CORBA.Object*.

When using Caffeine, the question of standardisation also needs to be considered. Caffeine is a vendor-specific feature supported only by Visibroker. When standard Java mapping objects are used, Caffeine is a powerful tool that simplifies the development process and is also CORBA compliant. But like many vendor specific features, there are also non-standard extensions that further ease implementation. However, the cost of using such features is the locking of oneself into a particular vendor and into one particular implementation. We believe that there is no correct answer as to which approach to take: there are many engineering trade-offs that need to be considered for specific situations.

5. Conclusions

The computing industry is developing new software technologies at a rapid rate. In order to apply these technologies effectively, research needs to be conducted to help understand and define the key characteristics of these technologies, the engineering tradeoffs that need to be considered, and the applicability of development approaches. This report argues that the use of technology experiments can help address these aspects by undertaking controlled case studies in particular application domains of interest. The results of this work can then provide practitioners with the information that is required to effectively use the technologies and can provide technology developers with insights into how the technologies might best evolve.

The Java/CORBA technology experiment discussed in this report has helped raise a number of issues that need to be considered if Java and CORBA are to provide the basis for application development. These include architectural considerations, support for component-based development, technolology selection issues, design approaches, and the need for effective tool support.

Although this work has been successful in terms of the original objectives, there are a range of other issues that need to be addressed in subsequent technology experiments. For example, this experiment considered the use of Java and CORBA technologies but did not make comparisons with other competing technologies such as Microsoft's ActiveX and DCOM. Moreover, there is a need to consider how the technologies might co-exist if used in the same application. Issues of scalability must also be considered. For example, in terms of the Attribute Cafè experiment, a follow-on experiment needs to be conducted with a more sophisticated client interface and a wider range of sources on a range of different machines (e.g. Notes databases, OO databases). Investigation of Interoperability between products of different vendors was beyond the scope of the experiment although it could be an important issue to resolve. Consideration also needs to be given as to how the results of particular experiments might relate to other domains (e.g. Defence Command and Control Information Systems), and the types of additional experiments that would need to be conducted for these larger and more complex applications.

6. References

ASTI (1998). *Graphical Designer Pro Product Information.* http://davinci2.csn.net/~jefscot/product.html.

Fenton, N., Pfleeger, S. L. and Glass, R. L. (1994). *Science and Substance: A Challenge to Software Engineers.* IEEE Software July: 86-95.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns - Elements of Reusable Object-Oriented Software.* Reading MA, Addison-Wesley.

IBM (1997). *IBM Component Broker.* http://www.software.ibm.com/ad/cb/.

Intersolv (1997). *Intersolv PVCS Series.* http://www.intersolv.com/products/scm.htm.

Iona (1997). *Orbix Product Overview.* http://www.iona.com/Products/Orbix/.

JavaSoft (1997). *Java Foundation Classes Overview.* http://www.javasoft.com/products/jfc/index.html.

Microgold (1997). *Microgold WithClass.* http://www.microgold.com/.

ObjectSpace *ObjectSpace technology.* http://www.objectspace.com.

OMG (1997). *Object Management Group.* http://www.omg.org.

Orfali, R. and Harkey, D. (1997). *Client/Server Programming with Java and CORBA,* Wiley Computer Publishing.

Rational (1997). *UML Resource Center.* http://www.rational.com/uml.

Rational (1997). *Visual Modelling With Rational Rose.* http://www.rational.com/products/rose.

Software, I. (1998). *Object Engineering Workbench for Java Product Information.* http://www.isg.de/OEW/Java/.

Sun (1997). *JavaIDL Overview.* http://www.javasoft.com/products/jdk/idl/docs/index.html.

Tichy, W. F. (1991). *RCS - A System for Version Control.* Indiana, Purdue University.

Vernik, R. J. (1996). *Visualisation and Description in Software Engineering.* Computer and Information Science. Adelaide, University of South Australia:232

Vinoski, S. (1997). *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments.* IEEE Communications Magazine(February 1997): 46-55.

Visigenic (1997). *Visibroker for Java 3.0 Programmer's Guide.* http://www.visigenic.com/techpubs/vbjava30/vbjpgmr.pdf.

Visigenic (1997). *Visigenic Software.* http://www.visigenic.com/.

Xerox (1997). *The Inter-Language Unification Project.* ftp://beta.xerox.com/pub/ilu/ilu.html.

Appendix A: UML Design Representations

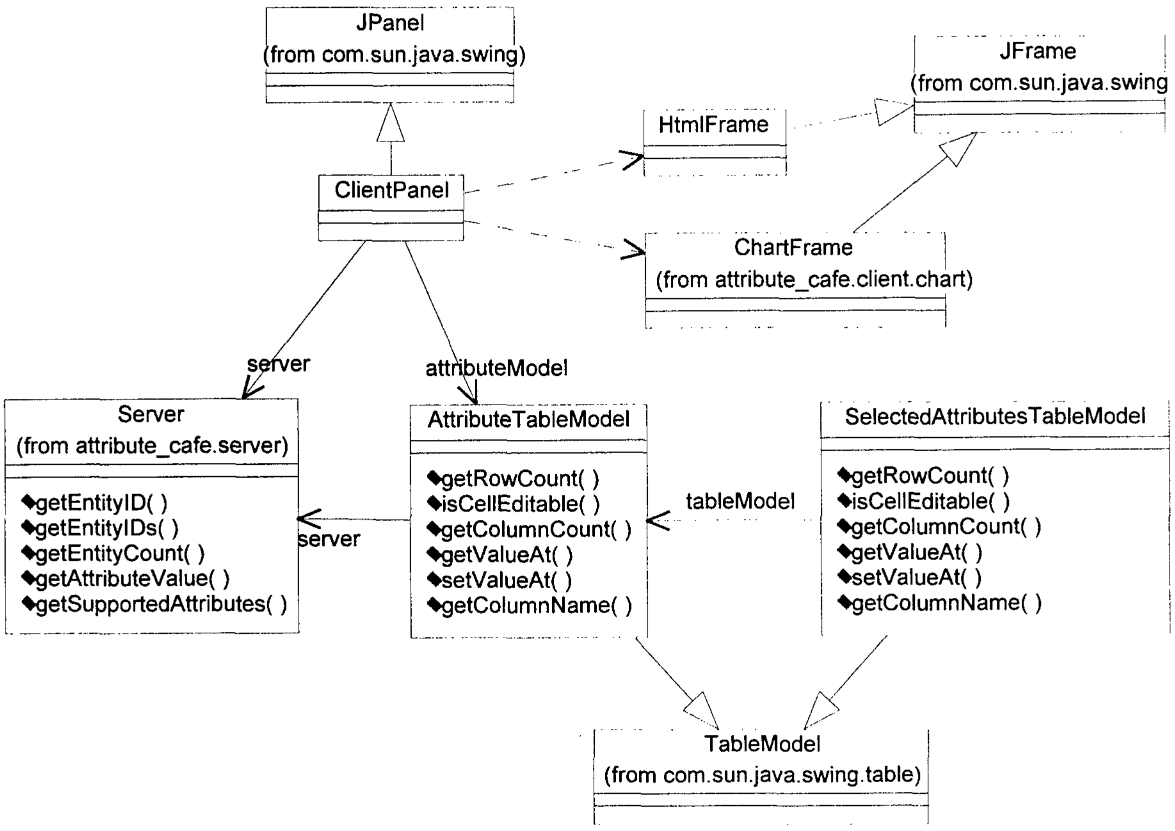


Figure A-1. Client UML diagram

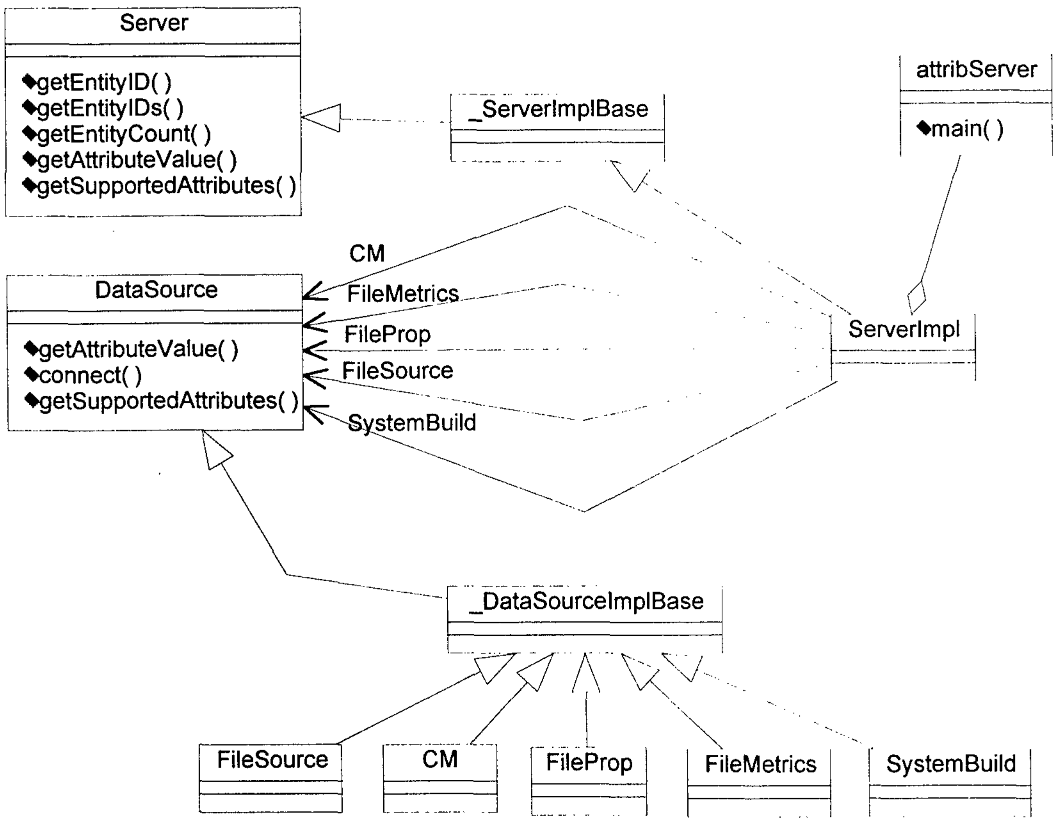


Figure A-2. Server UML diagram

Appendix B : Components Used

Component/Package	Function	Provider	JavaBean?
java.applet	Standard Java applet package	JavaSoft JDK	
java.awt	Standard Java windowing package	JavaSoft JDK	
java.net	Standard Java networking protocol package	JavaSoft JDK	
java.text	Standard Java text formatting package	JavaSoft JDK	
java.util	Standard Java utility package (container classes)	JavaSoft JDK	
XYLayout	Places controls at fixed locations in a window	JBuilder Control Library	✓
JCChart	Charting component	JClass	✓
Border	Styled border decoration for JFC Swing controls	JFC Swing	
JButton	Button control	JFC Swing	✓
JEditorPane	Displays HTML text in a text field	JFC Swing	✓
JOptionPane	Displays common messages/dialog boxes.	JFC Swing	✓
JPanel	Panel for arranging UI controls	JFC Swing	✓
JScrollPane	Adds automatic scrolling for UI components	JFC Swing	✓
JTabbedPane	Displays multiple pages of components one at a time using 'tabs' to switch between pages.	JFC Swing	✓
JTable	Table (spreadsheet) control	JFC Swing	✓
JToolbar	Tool bar control for floating/docked button bars.	JFC Swing	✓

DISTRIBUTION LIST

Attribute Café: A Java/CORBA Technology Experiment
(DSTO-TR-0722)

T.H. Toh, M.P Phillips, R.J. Vernik

	Number of Copies
AUSTRALIA	
DEFENCE ORGANISATION	
Task sponsor:	
FASDM	1
S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control sheet
Counsellor, Defence Science, Washington	Doc Control sheet
Scientific Adviser to MRDC Thailand	Doc Control sheet
Director General Scientific Advisers and Trials)	1 shared copy
Scientific Adviser - Policy and Command)	
Navy Scientific Adviser	1 copy of Doc Control sheet and 1 distribution list
Scientific Adviser - Army	Doc Control sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical & Maritime Research Laboratory	
Director	1
Electronics and Surveillance Research Laboratory	
Director	1
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control sheet
Head, C3I Systems Engineering Group	Doc Control sheet
Head, Information Warfare Studies Group	Doc Control sheet
Head, Software Engineering Group	Doc Control sheet
Head, Trusted Computer Systems Group	Doc Control sheet
Head, Advanced Computer Capabilities Group	Doc Control sheet
Head, Systems Simulation and Assessment Group	Doc Control sheet
Head, CCIS Interoperbility Lab	Doc Control sheet
Head, C3I Operational Analysis Group	Doc Control sheet

Head, Human Systems Integration Group	Doc Control sheet
Head, Y2000 Project	Doc Control Sheet
Head, C2 Australian Theatre Group	1
Head, Information Architectures Group	1
Head, Intelligence Systems Group	1
Head Command Support Systems Group	1
Head Information Management and Fusion Group	1
Task Manager	1
Author	1
Publications and Publicity Officer, ITD	1
DSTO Library and Archives	
Library Fishermens Bend	1
Library Maribyrnong	1
Library DSTOS	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control sheet
Forces Executive	
Director General Maritime Development,	Doc Control sheet
Director General Land Development,	Doc Control sheet
Director General C3I Development	1
Navy	
SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000.	Doc Control sheet
Army	
ABCA Office, G-1-34, Russell Offices, Canberra	4
Intelligence Program	
DGSTA, Defence Intelligence Organisation	1
Corporate Support Program (libraries)	
TRS Defence Regional Library, Canberra	1
Officer in Charge, Document Exchange Centre (DEC),	Doc Control sheet and 1 distribution list
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1
Universities and Colleges	
Australian Defence Force Academy Library	1
Head of Aerospace and Mechanical Engineering	1
Deakin University, Serials Section (mlist), Deakin University Library, Geelong, 3127	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1
DSTC (Distributed Systems Technology Centre), Queensland	1
Other Organisations	
NASA (Canberra)	1
AGPS	1
State Library of South Australia	1

Parliamentary Library, South Australia	1

OUTSIDE AUSTRALIA

Abstracting and Information Organisations	
INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts	1
Documents Librarian, The Center for Research Libraries, US	1
Information Exchange Agreement Partners	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	10
Total number of copies:	**64**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
2. TITLE Attribute Café: A Java/CORBA Technology Experiment	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) T.H. Toh, M.P Phillips and R.J. Vernik	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108 Australia

6a. DSTO NUMBER DSTO-TR-0722	6b. AR NUMBER AR-010-635	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE October 1998

8. FILE NUMBER N9505/15/113	9. TASK NUMBER 97/127	10. TASK SPONSOR DGCSS PMJP2030	11. NO. OF PAGES 26	12. NO. OF REFERENCES 29

13. DOWNGRADING/DELIMITING INSTRUCTIONS N/A	14. RELEASE AUTHORITY Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Java
CORBA (Computer Architecture)
Software Engineering

19. ABSTRACT

This report describes an experiment which was undertaken to consider a range of software engineering issues associated with the development of component-based distributed applications. The experiment focused on the use of Java and Common Object Request Broker (CORBA) technologies as used in the development of a software visualisation demonstator application (the Attribute Café) which extracts, integrates and presents software-related information. This experiment has raised a number of issues that need to be considered if Java and CORBA are to provide basis for application development. These include architectural considerations, support for component-based development, technolology selection issues, design approaches, and the need for effective tool support.

Page classification: UNCLASSIFIED